Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

Вариант 1

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ В.В. Тынченко

подпись, дата инициалы, фамилия

Студент КИ15–16Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

# Задача лабораторной работы

Разработать программу, реализующую метод Дэвидона-Флетчера-Пауэлла.

Найти безусловный экстремум функции, выбранной в соответствии с заданием, с использованием разработанной программы.

# Листинг программы

Point.cs

using System;

namespace ConsoleApp1

{

class Point

{

public double X { get; set; }

public double Y { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="x"></param>

/// <param name="y"></param>

public Point(double x, double y)

{

X = x;

Y = y;

}

public Point(Matrix m)

{

if (m.N != 2 && m.M!= 1)

return;

X = m.Element[0, 0];

Y = m.Element[1, 0];

}

/// <summary>

/// Длина

/// </summary>

/// <returns></returns>

public double GetLength()

{

return Math.Sqrt(X \* X + Y \* Y);

}

/// <summary>

/// Сложение

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static Point operator +(Point p1, Point p2)

{

return new Point(p1.X + p2.X, p1.Y + p2.Y);

}

/// <summary>

/// Вычитание

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static Point operator -(Point p1, Point p2)

{

return new Point(p1.X - p2.X, p1.Y - p2.Y);

}

/// <summary>

/// Скалярное произведение

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static double operator \*(Point p1, Point p2)

{

return p1.X \* p2.X + p1.Y \* p2.Y;

}

/// <summary>

/// Произведение на число

/// </summary>

/// <param name="p1"></param>

/// <param name="c"></param>

/// <returns></returns>

public static Point operator \*(Point p1, double c)

{

return new Point (p1.X \* c, p1.Y \* c);

}

public static Point operator \*(double c, Point p1)

{

return new Point(c \* p1.X, c \* p1.Y);

}

/// <summary>

/// Деление на число

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <returns></returns>

public static Point operator /(double c, Point p1)

{

return new Point(c / p1.X, c / p1.Y);

}

public static Point operator /(Point p1, double c)

{

return new Point(p1.X / c, p1.Y / c);

}

}

}

Matrix.cs

using System.Collections.Generic;

namespace ConsoleApp1

{

class Matrix

{

public double[,] Element { get; set; }

public int N { get; set; }a

public int M { get; set; }

/// <summary>

/// Конструкторы

/// </summary>

/// <param name="points"></param>

public Matrix(List<Point> points)

{

N = points.Count;

M = 2; // для Point

Element = new double[N, M];

for (int i = 0; i < N; i++)

{

Element[i, 0] = points[i].X;

Element[i, 1] = points[i].Y;

}

}

public Matrix(Point[] points)

{

N = points.Length;

M = 2; // для Point

Element = new double[N, M];

for (int i = 0; i < N; i++)

{

Element[i, 0] = points[i].X;

Element[i, 1] = points[i].Y;

}

}

public Matrix(Point point)

{

N = 1;

M = 2;

Element = new double[N, M];

Element[0, 0] = point.X;

Element[0, 1] = point.Y;

}

public Matrix(int n = 2, int m = 2)

{

N = n;

M = m;

Element = new double[N, M];

}

/// <summary>

/// Транспонирование

/// </summary>

/// <param name="m1"></param>

/// <returns></returns>

static public Matrix Transpose(Matrix m1) // +

{

Matrix m = new Matrix(m1.M, m1.N);

for (int i = 0; i < m.N; i++)

for (int j = 0; j < m.M; j++)

m.Element[i, j] = m1.Element[j, i];

return m;

}

/// <summary>

/// Умножение

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator \*(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] \* m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 \* m2.Element[0, 0];

return m;

}

else if (m1.M != m2.N)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

return m;

}

public static Point operator \*(Matrix m1, Point p) // +

{

Matrix m2 = new Matrix(p);

m2 = Transpose(m2); // Транспонируем

if (m1.M != m2.N)

return null;

Matrix m = new Matrix(m1.N, m2.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

p = new Point(m);

return p;

}

public static Point operator \*(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.M != m2.N)

return null;

Matrix m = new Matrix(m1.N, m2.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* m2.Element[k, j];

p = new Point(m);

return p;

}

public static Matrix operator \*(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = m1.Element[i, j] \* c;

return m;

}

public static Matrix operator \*(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = c \* m1.Element[i, j];

return m;

}

/// <summary>

/// Деление

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator /(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] / m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 / m2.Element[0, 0];

return m;

}

else if (m1.M != m2.N)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.M; j++)

for (int k = 0; k < m2.N; k++)

m.Element[i, j] += m1.Element[i, k] \* (1 / m2.Element[k, j]);

return m;

}

public static Matrix operator /(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = m1.Element[i, j] / c;

return m;

}

public static Matrix operator /(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N, m1.M);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.N; j++)

m.Element[i, j] = c / m1.Element[i, j];

return m;

}

/// <summary>

/// Вычитание

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator -(Matrix m1, Matrix m2) // +

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] - m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 - m2.Element[0, 0];

return m;

}

else if (m1.N != m2.N && m1.M != m2.M)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Matrix m1, Point p)

{

Matrix m2 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] - m2.Element[i, j];

return m;

}

public static Matrix operator -(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = m1.Element[i, j] - c;

return m;

}

public static Matrix operator -(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = c - m1.Element[i, j];

return m;

}

/// <summary>

/// Сложение

/// </summary>

/// <param name="m1"></param>

/// <param name="m2"></param>

/// <returns></returns>

public static Matrix operator +(Matrix m1, Matrix m2)

{

Matrix m = new Matrix(m1.N, m2.M);

if (m1.N == 1 && m1.M == 1)

{

m = m1.Element[0, 0] + m2;

return m;

}

else if (m2.N == 1 && m2.M == 1)

{

m = m1 + m2.Element[0, 0];

return m;

}

else if (m1.N != m2.N && m1.M != m2.M)

return null;

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Matrix m1, Point p)

{

Matrix m2 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Point p, Matrix m2)

{

Matrix m1 = new Matrix(p);

if (m1.N != m2.N && m1.M != m2.M)

return null;

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m2.N; j++)

m.Element[i, j] = m1.Element[i, j] + m2.Element[i, j];

return m;

}

public static Matrix operator +(Matrix m1, double c)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = m1.Element[i, j] + c;

return m;

}

public static Matrix operator +(double c, Matrix m1)

{

Matrix m = new Matrix(m1.N);

for (int i = 0; i < m1.N; i++)

for (int j = 0; j < m1.M; j++)

m.Element[i, j] = c + m1.Element[i, j];

return m;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using MathNet.Numerics;

namespace ConsoleApp1

{

class Program

{

/// <summary>

/// Функция

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static double Func(Point p) => Math.Pow(p.X - 2, 4) + Math.Pow(p.X - 2 \* p.Y, 2);

//2 \* Math.Pow(p.X, 2) + p.X \* p.Y + Math.Pow(p.Y, 2);

/// <summary>

/// Градиент функции в точке

/// </summary>

/// <param name="p"></param>

/// <returns></returns>

static Point GradFunc(Point p)

{

double dx = 0.001;

double dy = 0.001;

double derX = (Func(new Point(p.X + dx, p.Y)) - Func(p)) / dx;

double derY = (Func(new Point(p.X, p.Y + dy)) - Func(p)) / dy;

return new Point(derX, derY);

}

/// <summary>

/// Функция для поиска экстремума

/// </summary>

/// <param name="p"></param>

/// <param name="t"></param>

/// <param name="d"></param>

/// <returns></returns>

static double FuncExtremum(Point p, double t, Point d) => Math.Pow((p.X + t \* d.X) - 2, 4) + Math.Pow((p.X + t \* d.X) - 2 \* (p.Y + t \* d.Y), 2);

//2 \* Math.Pow(p.X + t \* d.X, 2) + (p.X + t \* d.X) \* (p.Y + t \* d.Y) + Math.Pow(p.Y + t \* d.Y, 2);

/// <summary>

/// Поиск экстремума методом одномерной минимизации (золотого сечения)

/// </summary>

/// <param name="p"></param>

/// <param name="d"></param>

/// <returns></returns>

static double FindExtremum(Point p, Point d)

{

double a = p.X + 100; // Левая граница по X начальной точки

double b = p.X - 100; // Правая граница по X начальной точки

double eps = 0.001;

double x = 0;

double y = 0;

double Fy = 0;

double z = 0;

double Fz = 0;

double length = 0;

int k = 0;

// Step 1

length = b - a;

// Step 2

k = 0;

// Step 3

y = a + ((3 - Math.Sqrt(5)) / 2) \* (b - a);

z = a + b - y;

while (true)

{

// Step 4

Fy = FuncExtremum(p, y, d);

Fz = FuncExtremum(p, z, d);

// Step 5

if (Fy <= Fz)

{

// a = a;

b = z;

z = y;

y = a + b - y;

}

else

{

a = y;

// b = b;

y = z;

z = a + b - z;

}

// Step 6

double delta = Math.Abs(a - b);

if (delta <= eps)

{

x = (a + b) / 2;

break;

}

else

{

k++;

}

}

return x;

}

/// <summary>

/// Вывести результат в консоль

/// </summary>

/// <param name="resultPoint"></param>

/// <param name="steps"></param>

static void PrintResult(Point resultPoint, int steps)

{

double result = Func(resultPoint);

result = Func(new Point(2, 1));

Console.WriteLine("X = " + resultPoint.X + "; Y = " + resultPoint.Y);

Console.WriteLine("F(x,y) = {0:F10}", result);

Console.WriteLine("Точность: {0:F10}", Math.Abs(Func(new Point(2, 1)) - result));

Console.WriteLine("Количество шагов: {0}", steps);

}

/// <summary>

/// Метод Дэвидона-Флетчера-Пауэлла

/// </summary>

/// <param name="x0"></param>

/// <param name="eps"></param>

static void Run(Point x0, double eps1 = 0.1, double eps2 = 0.15, int M = 10)

{

int reiteration = 0;

List<Point> x = new List<Point>();

x.Add(x0);

List<Point> d\_x = new List<Point>();

d\_x.Add(null);

List<Point> d\_g = new List<Point>();

d\_g.Add(null);

List<Matrix> A = new List<Matrix>();

List<Matrix> A\_c = new List<Matrix>();

List<Point> d = new List<Point>();

List<double> t = new List<double>();

// Шаг 1

Point gradF = GradFunc(x0); // !!! ненужная операция

// Шаг 2

int k = 0;

A.Add(new Matrix(new Point[] { new Point(1, 0), new Point(0, 1) })); // A[0]

while (true)

{

// Шаг 3

gradF = GradFunc(x[k]);

// Шаг 4

if (gradF.GetLength() < eps1)

{

// а

PrintResult(x[k], k);

return;

}

// Шаг 5

// а

if (k >= M)

{

PrintResult(x[k], k);

return;

}

// б

if (k != 0)

{

// Шаг 6

d\_g.Add(null);

d\_g[k - 1] = GradFunc(x[k]) - GradFunc(x[k - 1]);

// Шаг 7

d\_x.Add(null);

d\_x[k - 1] = x[k] - x[k - 1];

// Шаг 8

Matrix m1 = Matrix.Transpose(new Matrix(d\_x[k - 1])) \* new Matrix(d\_x[k - 1]);

Matrix m2 = new Matrix(d\_x[k - 1]) \* Matrix.Transpose(new Matrix(d\_g[k - 1]));

Matrix m3 = A[k - 1] \* Matrix.Transpose(new Matrix(d\_g[k - 1])) \* new Matrix(d\_g[k - 1]) \* A[k - 1];

Matrix m4 = new Matrix(d\_g[k - 1]) \* A[k - 1] \* Matrix.Transpose(new Matrix(d\_g[k - 1]));

Matrix m5 = m1 / m2;

Matrix m6 = m3 / m4;

Matrix m7 = m5 - m6;

A\_c.Add(null);

A\_c[k - 1] = m7;

// Шаг 9

A.Add(null);

A[k] = A[k - 1] + A\_c[k - 1];

}

// Шаг 10

d.Add(null);

d[k] = -1 \* A[k] \* gradF;

// Шаг 11

t.Add(0);

t[k] = FindExtremum(x[k], d[k]); // !!! не сказано, что подставлять d[k]

// Шаг 12

x.Add(null);

x[k + 1] = x[k] + t[k] \* d[k]; // !!! тут тоже можно заменить на d[k]

// Шаг 13

// а

if ((x[k + 1] - x[k]).GetLength() < eps2 && Math.Abs(Func(x[k + 1]) - Func(x[k])) < eps2)

{

reiteration++;

if (reiteration == 2) // j - 1 тоже соответствовало данному условию

{

PrintResult(x[k + 1], k);

return;

}

}

// б

else

{

reiteration = 0;

k++;

continue;

}

}

}

/// <summary>

/// Главная функция

/// </summary>

/// <param name="args"></param>

static void Main(string[] args)

{

Stopwatch sw = new Stopwatch();

Console.WriteLine("Тест 1");

Console.WriteLine("Берем стандартные параметры");

sw.Reset();

sw.Start();

Run(new Point(10, 10));

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 2");

Console.WriteLine("Увеличим eps1");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.5);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 3");

Console.WriteLine("Уменьшим eps1");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.01);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 4");

Console.WriteLine("Увеличим eps2");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 0.5);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 5");

Console.WriteLine("Уменьшим eps2");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 0.01);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 6");

Console.WriteLine("Уменьшим M");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 0.15, 4);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.WriteLine("Тест 7");

Console.WriteLine("Увеличим M");

sw.Reset();

sw.Start();

Run(new Point(10, 10), 0.1, 0.15, 20);

sw.Stop();

Console.WriteLine("Скорость алгоритма: {0} ms\n", sw.ElapsedMilliseconds);

Console.ReadKey();

}

}

}

# Скриншоты результата выполнения программы

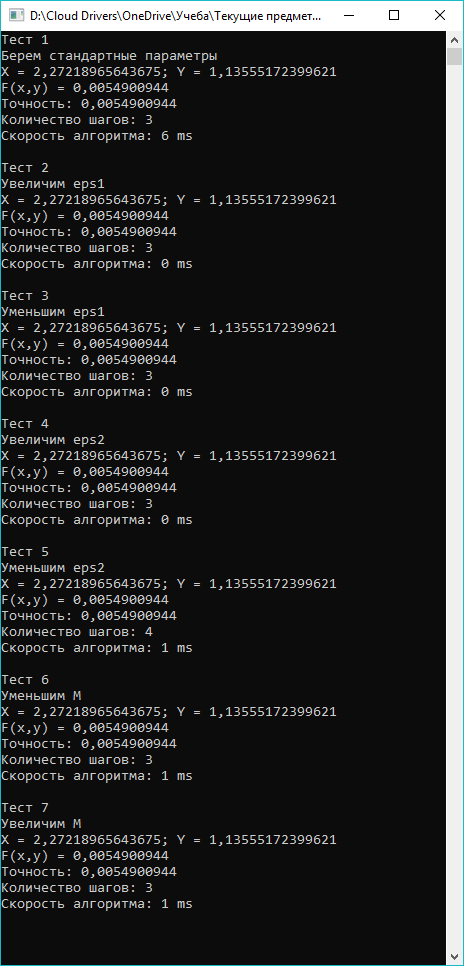


Рисунок 1 – Пример работы программы

# Выводы

В результате проделанной работы была разработана программа, которая выполнила поиск безусловного экстремума (минимума) заданной функции методом Дэвидона-Флетчера-Пауэлла (далее – ДФП). В ходе тестов были установлены следующие зависимости при изменениях параметров (таблица 1): при изменении каждого параметра результаты тестирования одинаковы. Это происходит вследствие утверждения о том, что алгоритм метода ДФП в применении к квадратичной функции с положительно определенной матрицей Гессе обеспечивает отыскание минимума не более чем за N шагов.

Таблица 1 – Исследование алгоритма путем изменения параметров

|  |  |
| --- | --- |
| Тест 1  Берем стандартные параметры  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 6 ms | Тест 2  Увеличим eps1  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 0 ms |
| Тест 3  Уменьшим eps1  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 0 ms | Тест 4  Увеличим eps2  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 0 ms |
| Тест 5  Уменьшим eps2  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 4  Скорость алгоритма: 1 ms | Тест 6  Уменьшим M  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 1 ms |
| Тест 7  Увеличим M  X = 2,27218965643675; Y = 1,13555172399621  F(x,y) = 0,0054900944  Точность: 0,0054900944  Количество шагов: 3  Скорость алгоритма: 1 ms |  |

Также стоит сравнить данный алгоритм с методом покоординатного спуска и методом Флетчера-Ривса. Однако стоит отметить, что исследуемые функции различны в плане расположения точки минимума, но это не мешает сравнить точность. Результаты приведены в таблице 2:

Таблица 2 – Сравнение различных алгоритмов при значениях x0 = (10,10) eps1=0.1, eps2 = 0.15, M = 10)

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод** | **Время** | **Точность результата** | **Шаги** |
| Покоординатного спуска | 3 мс | 0.0011 | 8 |
| Флетчера-Ривса | 3 мс | 0.00053 | 2 |
| Дэвидона-Флетчера-Пауэлла | 6 мс | 0.00549 | 3 |

Примечательно, что ДФП показал наихудшую точность результата среди всех методов первого порядка. Время тоже было затрачено больше, чем в других методах в связи с использованием матриц и вычислений, связанных с ними. Однако для квадратичных функций он является лучшим в плане того, что при применении для минимизации квадратичной функции N переменных он сходится не более чем за N итераций. Это весьма мощная оптимизационная процедура, очень эффективная при оптимизации большинства функций независимо от того, квадратичны они или нет. Среди всех методов наилучшие результаты показал метод Флетчера-Ривса, однако ДФП для большинства функций в теории будет эффективнее, однако на данном примере так не произошло, что говорит о том, что нет универсальных методов для решения любой задачи.